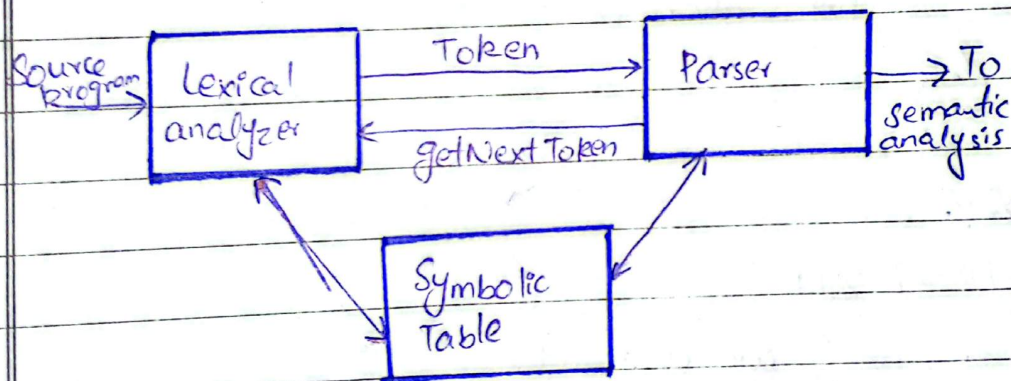


Question #01

Part #01

Explain lexical analyzer tool:- (Lex/flex)

Lex/flex is a tool that generates lexical analyzer (Scanners) that convert input character streams into tokens.



Structure of lex program

declaration

% %

translation rules

% %

auxiliary functions

Date

Translation rules each of the form:-

Pattern {Action}

Sample Lex program:-

```
% {
```

```
# include <stdio.h>
```

```
int word_count = 0;
```

```
int line_count = 0;
```

```
int char_count = 0;
```

```
% }
```

```
%.
```

```
[a-zA-Z]+
```

```
{ word_count++;
```

```
char_count += yylen; }
```

```
\n
```

```
{ line_count++;
```

```
char_count++; }
```

```
[\t]
```

```
{ char_count++; }
```

```
.
```

M
Mujahid

Date _____

```
    } char-count ++}  
    %i.%i.%i
```

```
int main()
```

```
{
```

```
    yylex();
```

```
    printf("\n===Summary===\n");
```

```
    printf("Lines: %d\n", line-count);
```

```
    printf("Words: %d\n", word-count);
```

```
    printf("Characters: %d\n", char-count);
```

```
    return 0;
```

```
}
```

```
int yywrap()
```

```
{
```

```
    return 1;
```

```
}
```

Part # 02

Input file:-

```
void main()
```

```
{
```

```
    int sum = 0;
```

```
    for (int j=0; j<10; j+=j+1)
```

```
    {
```

```
        Sum = Sum + j + 10.43 + 344 +
```

```
        45.34E-4 + 43.0 + 7.34;
```

```
    }
```

```
}
```

mylexer.1 File

```
%option noyywrap
```

```
%{
```

```
    #include <stdio.h>
```

```
    #include <stdlib.h>
```

```
    #include <string.h>
```

```
    int keyword-counter = 0;
```

int id_counter = 0;
 int operator_counter = 0;
 int number_counter = 0;
 int string_counter = 0;
 int lb_counter = 0;
 int rb_counter = 0;
 int lcb_counter = 0;
 int rcb_counter = 0;
 int semi_counter = 0;
 int lsqb_counter = 0;
 int rsqb_counter = 0;

% }

digit [0-9]

letter [a-zA-Z]

identifier {letter} + ({letter} | {digit})*

number {digit}+ ({digit}+)? ([eE][+-])?
 {digit}+)?

whitespace [\t \n]

% %

"int"

{ keyword_counter++;

printf("<keyword, %d> \t \t int \n",

keyword_counter);

M
Mujahid

"void"

```
{ keyword-counter++;  
printf("<keyword, %d> \t\t void \n",  
keyword-counter);  
}
```

"main"

```
{ keyword-counter++;  
printf("<keyword, %d> \t\t main \n",  
keyword-counter);  
}
```

"for"

```
{  
keyword-counter++;  
printf("<keyword, %d> \t\t for \n",  
keyword-counter);  
}
```

"return"

```
{  
keyword-counter++;  
printf("<keyword, %d> \t\t return  
\n", keyword-counter);  
}
```

{number}

{

number-counter++;

printf("<number, %d>|t|t%s",
number-counter, yytext);

}

{identifier}

{

id-counter++;

printf("<id, %d>|t|t%s\n",
id-counter, yytext);

}

"="

{

operator-counter++;

printf("<operator, %d>|t|t=|n",
operator-counter); }

"+"

{

operator-counter++;

printf("<operator, %d>|t|t+|n",
operator-counter);

}

"-"

```
{ operator-counter++;  
  printf("operator, %d > |t|t - |n",  
    operator-counter);  
}
```

"*"

```
{ operator-counter++;  
  printf("operator, %d > |t|t * |n",  
    operator-counter);  
}
```

"/"

```
{ operator-counter++;  
  printf("operator, %d > |t|t / |n",  
    operator-counter);  
}
```

"<"

```
{ operator-counter++;  
  printf("operator, %d > |t|t < |n",  
    operator-counter);  
}
```

">"

```
{  
  operator-counter++;  
  printf("operator, %d > |t|t > |n",  
    operator-counter);  
}
```


22

```
{ printf("comma,%d\t%d\t%d\t%d\n"),
```

{white space}

9/10 ignore whitespace, 1/3

```

2
3
printf ("UNKNOWN : %s\n",
yytext );
}

```

$$\frac{1}{10} \div \frac{1}{10}$$

int main()

```

    printf ("Token %t %t %t %lexem %n");
    gylex();
    printf ("\nCompleted\n");
    return 0;
}

```

Output file:-

Token	Lexem
<keyword, 1>	void
<keyword, 2>	main
<lb, 1>	(
<rb, 1>	)
<lb, 1>	{
<keyword, 3>	int
<id, 1>	sum
<operator, 1>	=
<number, 1>	0
<semi, 1>	;
<keyword, 4>	for
<lb, 2>	(
<keyword, 5>	int
<id, 2>	i
<operator, 2>	=
<number, 2>	0
<semi, 2>	;
<id, 3>	j
<operator, 3>	<
<number, 3>	10
<semi, 3>	;
<id, 4>	j
<operator, 4>	=

Date

<id, 5>	}
<operator, 5>	+
<number, 4>	1
<r b, 2>	)
<lcb, 2>	}
<id, 6>	Sum
<operator, 6>	=
<id, 7>	Sum
<operator, 7>	+
<id, 8>	j
<operator, 8>	+
<number, 5>	10.43
<operator, 9>	+
<number, 6>	344
<operator, 10>	+
<number, 7>	45.84E-4
<operator, 11>	+
<number, 8>	430
<operator, 12>	+
<number, 9>	734
<sem, 4>	;
<r cb, 1>	}
<r cb, 2>	}

Completed

Date

Compiler Construction

Tools:-

Scanner Generator:-

Generate the lexical analyzer, which reads the source code and group characters into meaningful sequences called tokens.

Example:- Lex, Flex.

Parser Generators:-

Generate the syntax analyzer, which takes the stream of tokens from the lexer and checks if they form a valid structure according to the grammar. It often builds a parse tree or syntax tree.

Example:- Yacc, GNU Bison, CUP



Mujahid

Syntax-Directed Tools:-

Handle attribute grammars and enable format spec semantic analysis of code structure.

Example:-

stablecc

Code-generators :- Perform advanced code optimization to provide best performance and efficiency.

Example:-

LLVM

JIT Compilers - Generate machine code dynamically at runtime for adaptive high speed execution.

Running

mylexer.l $\rightarrow$ Lex compiler $\rightarrow$ lex.yy.c

lex.yy.c $\rightarrow$ C-Compiler $\rightarrow$ mylexer.exe

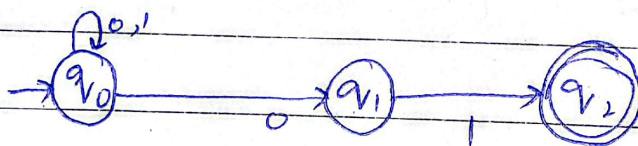
input.txt $\rightarrow$ mylexer.exe $\rightarrow$ Sequence of tokens

Commands to run:-

- flex mylexer.l
- gcc lex.yy.c -o mylexer
- mylexer.exe < input.txt

Question # 02

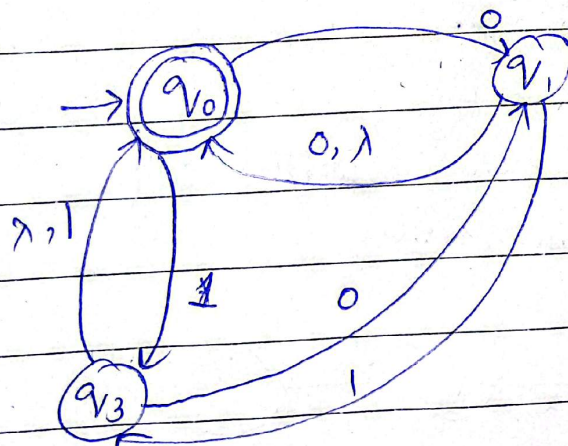
(i) $L_1 = \{w/w \text{ ends with } 01\}$,
with 3 states:-



(2) $L2 = \{w/w \text{ contains an even number of 0's or even no of 1's}\}$, with 3 states:-

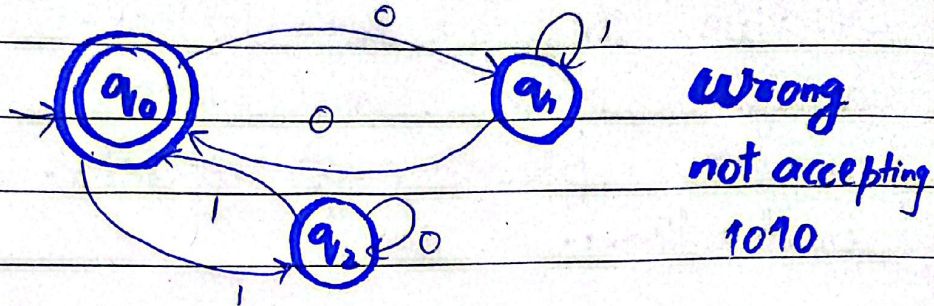
It is impossible because tracking the independent parties of both 0s and 1s requires four distinct memory combinations (2×2), and merging any of them in a 3-state machine would cause it to incorrectly accept strings with an odd number of both characters, like 01.

(1)

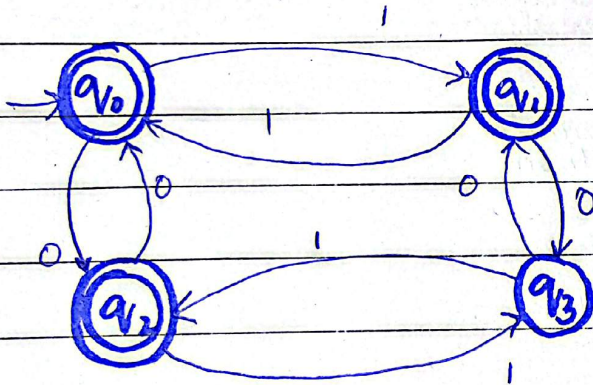


Wrong
accepting
01 and
10 etc.

(2)



∴ With 4-States:-



Date

Question #03

Describe the role of lexical-analysis phase and syntax-analysis phase of compiler-constructions:-

Lexical vs Syntax analysis

Aspect	Lexical analyzer	Syntax analyzer
Input	Character stream	Token stream
Output	Token stream	Parse tree
Recognition	Regular expressions	Context free grammar
Errors detection	Invalid characters, unterminated strings.	Missing semi-colons, mismatched braces, incorred structure.

Source Program

Lexical analysis

Input: character stream

Output: Token stream

Functions:-

- Remove white spaces and comments.
- Generate Tokens
- Handle errors like invalid characters.

Token stream

Syntax analysis

Input: Token stream

Output: Parse Tree

Functions:

- Check token sequence against grammar rules
- Construct parse tree
- Report error messages

Parse Tree

Question # 4:-

$$\begin{aligned}
 S &\rightarrow aS/bS/cS/dS/e/f/g/\epsilon Ag \\
 A &\rightarrow h/u/\lambda/Sab/SBb/SAh \\
 B &\rightarrow bBg/bbB/Bfc/\lambda
 \end{aligned}$$

Remove Left recursion:-

Given:-

$$\begin{aligned}
 S &\rightarrow aS/bS/cS/dS/e/f/g/\epsilon Ag \\
 A &\rightarrow h/u/\lambda/Sab/SBb/SAh \\
 B &\rightarrow bBg/bbB/Bfc/\lambda
 \end{aligned}$$

Removing Left Recursion:-

$$\begin{aligned}
 S &\rightarrow aS/bS/cS/dS/e/f/g/\epsilon Ag \\
 A &\rightarrow h/u/\lambda/Sab/SBb/SAh \\
 B &\rightarrow bBgB'/bbBB'/\lambda B' \\
 B' &\rightarrow fcb'/\lambda
 \end{aligned}$$

$$\begin{aligned}
 S &\rightarrow aS/bS/cS/dS/e/f/g/\epsilon Ag \\
 A &\rightarrow h/u/\lambda/Sab/SBb/SAh \\
 B &\rightarrow bBgB'/bbBB'/B' \\
 B' &\rightarrow fcb'/\lambda
 \end{aligned}$$

First

Follow

S {a, b, c, d, e, f, g, v}
A {b, v, λ, a, b, c, d, f, g, v}
B {b, f, λ}
B' {f, λ}

{\$, a, b, f, c, d, g, h, v, u}
 {g, h}
 {b, f, g}
 {b, f, g}

M	a	b	c	d	e	f	g	h	v	u	\$
S	$S \rightarrow as$	$S \rightarrow bS$	$S \rightarrow cS$	$S \rightarrow dS$	$S \rightarrow e$	$S \rightarrow f$	$S \rightarrow g$		$S \rightarrow v$		
A	$A \rightarrow Sab$ $A \rightarrow SBb$ $A \rightarrow SAh$	$A \rightarrow Sab$ $A \rightarrow SBb$ $A \rightarrow SAh$	$A \rightarrow Sab$ $A \rightarrow SBb$ $A \rightarrow SAh$	$A \rightarrow Sab$ $A \rightarrow SBb$ $A \rightarrow SAh$	$A \rightarrow Sab$ $A \rightarrow SBb$ $A \rightarrow SAh$	$A \rightarrow Sab$ $A \rightarrow SBb$ $A \rightarrow SAh$	$A \rightarrow Sab$ $A \rightarrow SBb$ $A \rightarrow SAh$ $A \rightarrow \lambda$	$A \rightarrow h$ $A \rightarrow \lambda$	$A \rightarrow Sab$ $A \rightarrow SBb$ $A \rightarrow SAh$	$A \rightarrow u$	
B	$B \rightarrow bB$ $B \rightarrow bB'$	$B \rightarrow bB$ $B \rightarrow bB'$ $B \rightarrow B'$				$B \rightarrow B'$ $B \rightarrow B'$	$B \rightarrow B'$				
B'		$B' \rightarrow \lambda$				$B' \rightarrow f$ $B' \rightarrow \lambda$	$B' \rightarrow \lambda$				